

GOING SERVERLESS: AN INTRODUCTION TO AWS LAMBDA

BY ALEX WILSON

A LITTLE ABOUT ME

Name: Alex Wilson

Role: Engineering at T. Rowe Price

Technology stack:

- Front end:
 - Angular JavaScript Framework
- Backend:
 - AWS Dynamo DB
 - Java Spring RESTful Services



I also like to code in React, ensure website accessibility, work on open source projects, and give talks and lectures like this one!

You can find me by going to my website: CodeByAlex.com

Or follow me on Twitter and Github: [@CodeByAlex](https://twitter.com/CodeByAlex)

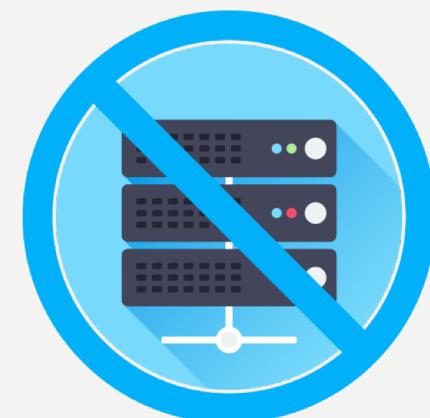
WHAT IS SERVERLESS COMPUTING?

Serverless computing allows you to build and run applications and services without thinking about managing the servers yourself.

Your application still runs on servers, but all the server management is done by an outside entity.

Serverless computing saves you money because your server is not running at all times

Let's be real...the name is misleading



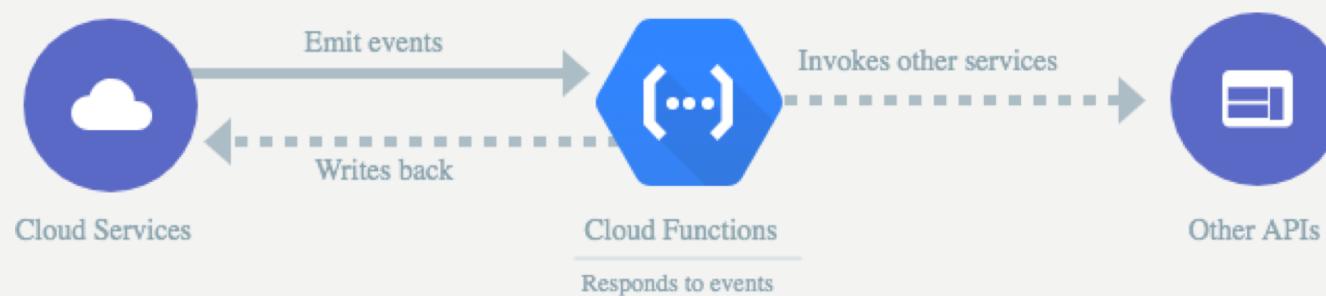
WHAT IS A SERVERLESS FUNCTION OR A FUNCTION AS A SERVICE (FAAS)?

A Serverless function is a single function that can be readily made available when you need it from anywhere in the world

The “server” suddenly appears when you need it, then disappears when you’re done with it.

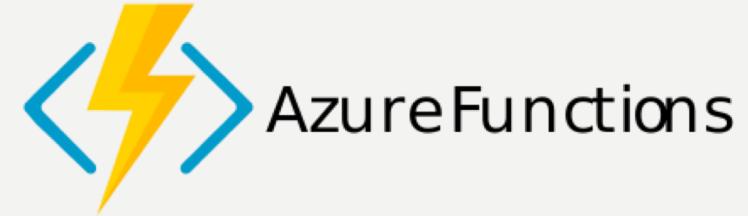
Now you can think in terms of functions instead of servers, and all your business logic can now live within these functions.

Allows developers to focus on their code rather than to think about the multi-threading or load-balancing of their servers. They trust the FaaS to handle all the resource management for them.



FUNCTIONS AS A SERVICE (FaaS) PLATFORMS

- AWS Lambda
- Azure Functions
- Google Cloud Functions
- IBM OpenWhisk
- Fission.io



WHAT IS AWS?

Amazon Web Services (AWS) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow.—
Amazon

It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.



AWS LAMBDA



The code you run on AWS Lambda is uploaded as a “Lambda function”.

Each function has associated configuration information, such as its name, description, entry point, and resource requirements.

Lambda functions can include libraries, even native ones.

You pay only for the compute time that you consume — there is no charge when your code is not running.

With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

HOW DO YOU TRIGGER AN AWS LAMBDA FUNCTION?

- An HTTP request using an API Gateway (What we are going to use today)
- When creating, removing, or updating a file in Amazon Simple Storage Service (S3), an event can be triggered to call a specified lambda function.
- A notification from AWS Simple Notification Service (SNS).

TO HAVE STATE...OR NOT TO HAVE STATE

For a function to have state means that the instance of that function has properties that you can refer back to and the values will remain attached to those properties there.

Keeping functions stateless enables AWS Lambda to rapidly launch as many copies of the function as needed to scale to the rate of incoming events.

While AWS Lambda's programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

WHAT CODING LANGUAGES RUN IN AN AWS LAMBDA FUNCTION?

- Java
- Node.js
- Python
- C#
- PowerShell
- Go
- Ruby
- Provides a Runtime API which allows you to use any additional programming languages to author your functions

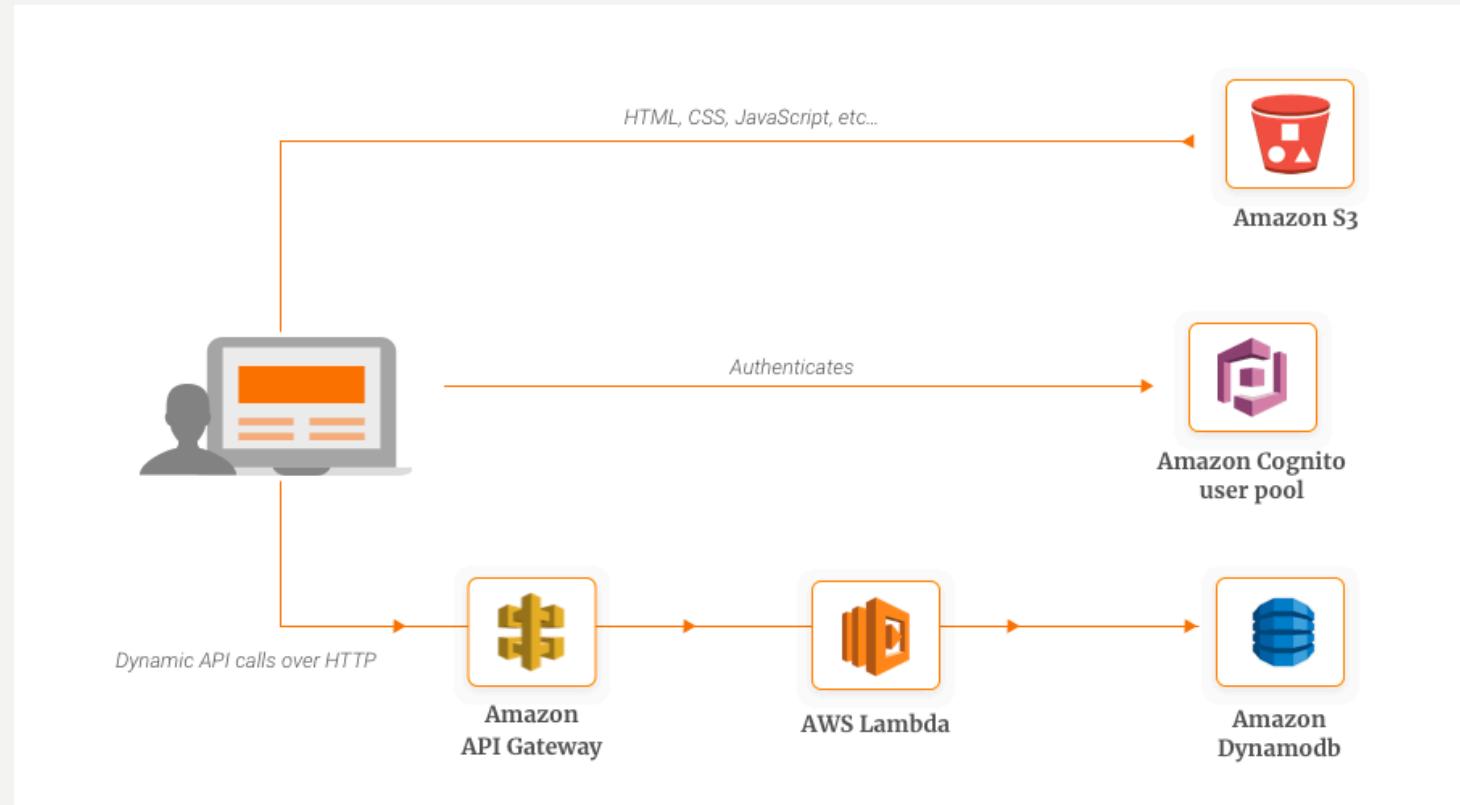
WHAT ARE THE USE CASES FOR AN AWS LAMBDA SERVERLESS FUNCTION?

- Chat bots: scaling automatically for peak demands
- Hackathon projects that need to be set up quickly with cloud based compute
- Create useful IOT functions that could be used with Google Home, Alexa, and other IOT devices
- Batch jobs scheduled tasks: Jobs that require intense parallel computation, IO or network access
- Continuous integration pipeline: The ability to remove the need for pre-provisioned hosts
- Send mass emails

In the next set of slides, we will dive deeper into some of these use cases

USE CASE #0: SERVERLESS WEBSITE EXAMPLE WITH AWS LAMBDA

- JavaScript in the browser exchanges the data from a backend API built through API Gateway and AWS Lambda.
- DynamoDB is a NoSQL database which is used for storing data through API's Lambda function.
- Amazon S3 is used for hosting the static website content like HTML, media files, CSS, JavaScript which acts as a front end in the user's browser.
- Amazon Cognito is used for user authentication and management with the help of secured backend API.

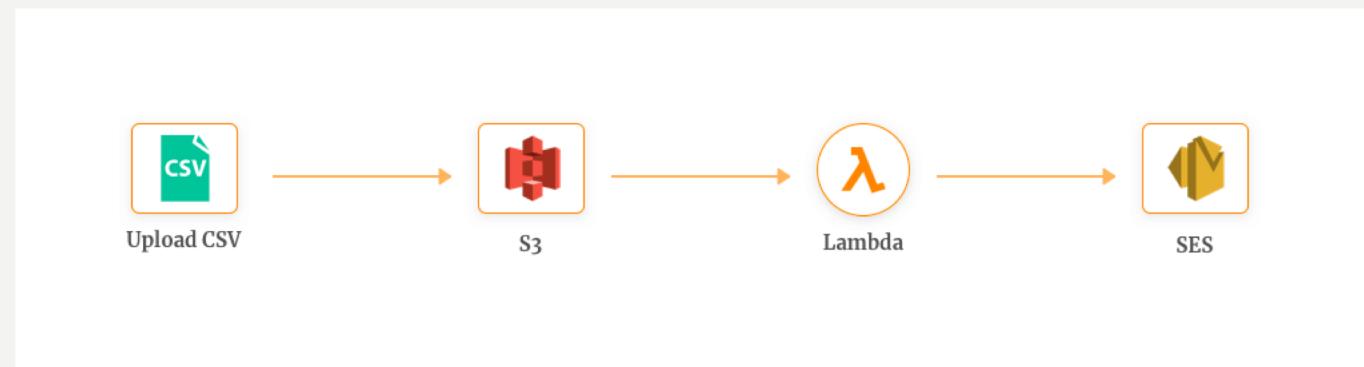


Use case from: <https://www.simform.com/serverless-examples-aws-lambda-use-cases/>

USE CASE #1: MASS EMAILING USING AWS LAMBDA & SES

With AWS Lambda and Simple Email Service SES, you can build a cost-effective and in-house serverless email platform. Along with S3 (where your mailing list will be stored) you can quickly send HTML or text-based emails to a large number of recipients.

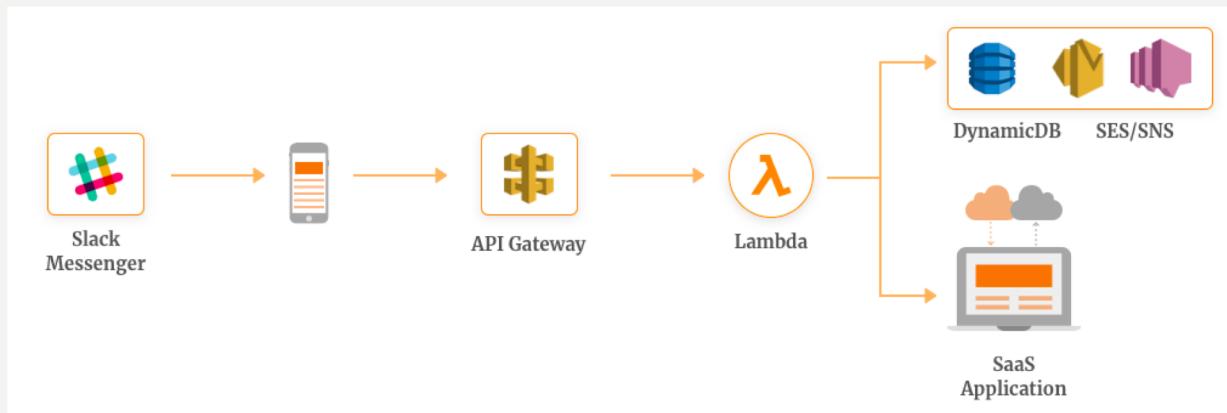
Whenever a user uploads a CSV file, it triggers an S3 event. This event triggers another Lambda function which imports the file into the database and will start sending email to all the addresses. For sending our scheduled newsletters, you can integrate it with CloudWatch Events.



Use case from: <https://www.simform.com/serverless-examples-aws-lambda-use-cases/>

USE CASE #2: AWS LAMBDA USE CASE FOR BUILDING SERVERLESS CHATBOT

- Building and running chatbots is not only time consuming but expensive also. Developers must provision, run and scale the infrastructural resources that run the chatbot code. However, with AWS Lambda you can run a scalable chatbot architecture.
- Enter your code logic to the Lambda function.
- Set up your code to trigger when user commands are sent to the bot. The commands are API requests (from Slack, Messenger, etc.) routed through API Gateway to Lambda function.
- Lambda runs only when it is commanded and hence using the resources when needed. You pay for the time it runs your code.



Use case from: <https://www.simform.com/serverless-examples-aws-lambda-use-cases/>

LET'S BUILD AN AWS LAMBDA FUNCTION!

- You will need:
 - An Amazon/AWS account
 - These slides to go at your own pace:
 - http://codebyalex.com/assets/AWS_Lambda_Lecture.pdf
- You will need to know (I will briefly go over both of these):
 - A coding language (today we will be using Node)
 - REST and RESTful API Methods
 - AWS Lambda Pricing information (we will be using the free tier)

NODE.JS: A BRIEF OVERVIEW

- A free open source server environment
- Runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Built on Chrome's JavaScript run-time for building fast and scalable network applications.
- Uses JavaScript on the server
- Asynchronous and event driven
- Single Threaded but highly scalable

* If you understand JavaScript and computer science basics, you should be well off for today's exercise

REST AND RESTFUL API METHODS

REST

- REST stands for Representational State Transfer
- Allows the transfer of data objects from one entity to another over HTTP
- Language-independent architectural style given that the language can make web-based requests using HTTP

RESTful API Methods

- GET – Used to read data from a data source
- PUT – Used to update and replace data to a data source
- POST – Used to create and save new data objects to a data source
- DELETE – Used to delete data from a data source

LAMBDA FUNCTION PRICING INFORMATION

Lambda Pricing Details

Lambda counts a **request** each time it starts executing in response to an event notification or invoke call, including test invokes from the console. You are charged for the total number of requests across all your functions.

Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 100ms. The price depends on the amount of memory you allocate to your function.

The Lambda free tier includes **1M free requests per month** and **400,000 GB-seconds of compute time per month**.

Free Tier

1M REQUESTS

per month

400,000 GB-SECONDS

of compute time per month.

The Lambda free tier does not automatically expire at the end of your 12 month AWS Free Tier term, but is available to both existing and new AWS customers indefinitely.

Requests

1M REQUESTS FREE

First 1M requests per month are free.

\$0.20 PER 1M REQUESTS THEREAFTER

\$0.0000002 per request.

Duration

400,000 GB-SECONDS PER MONTH FREE

First 400,000 GB-seconds per month, up to 3.2M seconds of compute time, are free.

\$0.00001667 FOR EVERY GB-SECOND USED THEREAFTER

The price depends on the amount of memory you allocate to your function.

* I am not liable for any accidental expenditures you might make when using AWS

API GATEWAY (USED TO CALL THE LAMBDA FUNCTION) PRICING

Free Tier

The Amazon API Gateway free tier includes **one million API calls** received for HTTP/REST APIs, and **one million messages and 750,000 connection minutes** for WebSocket APIs per month for **up to 12 months**. If you exceed this number of calls per month, you will be charged the API Gateway usage rates.

1M API CALLS RECEIVED | 1M MESSAGES | 750,000 CONNECTION MINUTES

per month

These free tier offers are only available to new AWS customers, and are available for 12 months following your AWS sign-up date. When your 12 month free usage term expires or if your application use exceeds the tiers, you simply pay standard, pay-as-you-go service rates.

[Sign up for the Free Tier](#)

* Free tier expires after 12 months of using AWS

* I am not liable for any accidental expenditures you might make when using AWS

LET'S BEGIN: GO TO THE AWS CONSOLE

AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.

X

Lambda
Run Code without Thinking about Servers

CodeBuild
Build and Test Code

IoT 1-Click
Trigger AWS Lambda functions from simple devices [View solution](#)

Get started with simple wizards and automated workflows.

Launch a virtual machine
With EC2
2-3 minutes 

Build a web app
With Elastic Beanstalk
6 minutes 

Build using virtual servers
With Lightsail
1-2 minutes 

Connect an IoT device
With AWS IoT
5 minutes 

Start a development project
With CodeStar
5 minutes 

Register a domain
With Route 53
3 minutes 

Access resources on the go
Access the Management Console using the AWS Console Mobile App. [Learn more](#)

Explore AWS

Scalable, Durable, Secure Backup & Restore with Amazon S3
Discover how customers are building backup & restore solutions on AWS that save money. [Learn more](#)

Amazon SageMaker
Machine learning for every developer and data scientist. [Learn more](#)

Visit AWS around the world at a Summit
AWS Global Summits bring the cloud computing community together to connect, collaborate, and learn about AWS. [Learn more](#)

Amazon RDS

CLICK “CREATE A FUNCTION”

COMPUTE

AWS Lambda

lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

Create a function

How it works

Run

Next: Lambda responds to events

```
1 exports.handler = (event, context, callback) => {  
2     // Succeed with the string "Hello world!"  
3     callback(null, 'Hello world!');  
4 };
```

CHOOSE AUTHOR FROM SCRATCH

Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

Author from scratch Start with a simple Hello World example.


Use a blueprint Build a Lambda application from sample code and configuration presets for common use cases.


Browse serverless app repository Deploy a sample Lambda application from the AWS Serverless Application Repository.


Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.

Permissions Info

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

► [Choose or create an execution role](#)

SET FUNCTION NAME AND CHOOSE NODEJS VERSION

Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

- Author from scratch** Start with a simple Hello World example.

- Use a blueprint** Build a Lambda application from sample code and configuration presets for common use cases.

- Browse serverless app repository** Deploy a sample Lambda application from the AWS Serverless Application Repository.


Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.

PERMISSIONS Info
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▶ [Choose or create an execution role](#)

CREATE A NEW ROLE WITH BASIC LAMBDA PERMISSIONS

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▼ Choose or create an execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

ⓘ Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

Lambda will create an execution role named demoFunction-role-ebql3nh2, with permission to upload logs to Amazon CloudWatch Logs.

[Cancel](#) [Create function](#)

Execution role – is an IAM entity that defines a set of permissions for making AWS service requests

CLICK “CREATE FUNCTION”

Basic information

Function name
Enter a name that describes the purpose of your function.

demoFunction

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Node.js 8.10

Permissions [Info](#)

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ Choose or create an execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role with basic Lambda permissions

ⓘ Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

Lambda will create an execution role named demoFunction-role-ebql3nh2, with permission to upload logs to Amazon CloudWatch Logs.

[Cancel](#) Create function

CONGRATS YOU'VE CREATED A LAMBDA FUNCTION! (KIND OF)

⌚ Congratulations! Your Lambda function "demoFunction" has been successfully created. You can now change its code and configuration. Choose Test to input a test event when you want to test your function. X

Lambda > Functions > demoFunction ARN - arn:aws:lambda:us-east-1:375912685630:function:demoFunction

demoFunction Throttle Qualifiers Actions Select a test event Test Save

Configuration Monitoring

Designer

Add triggers Choose a trigger from the list below to add it to your function.

- API Gateway
- AWS IoT
- Alexa Skills Kit
- Alexa Smart Home
- Application Load Balancer
- CloudFront

demoFunction Layers (0)

Add triggers from the list on the left

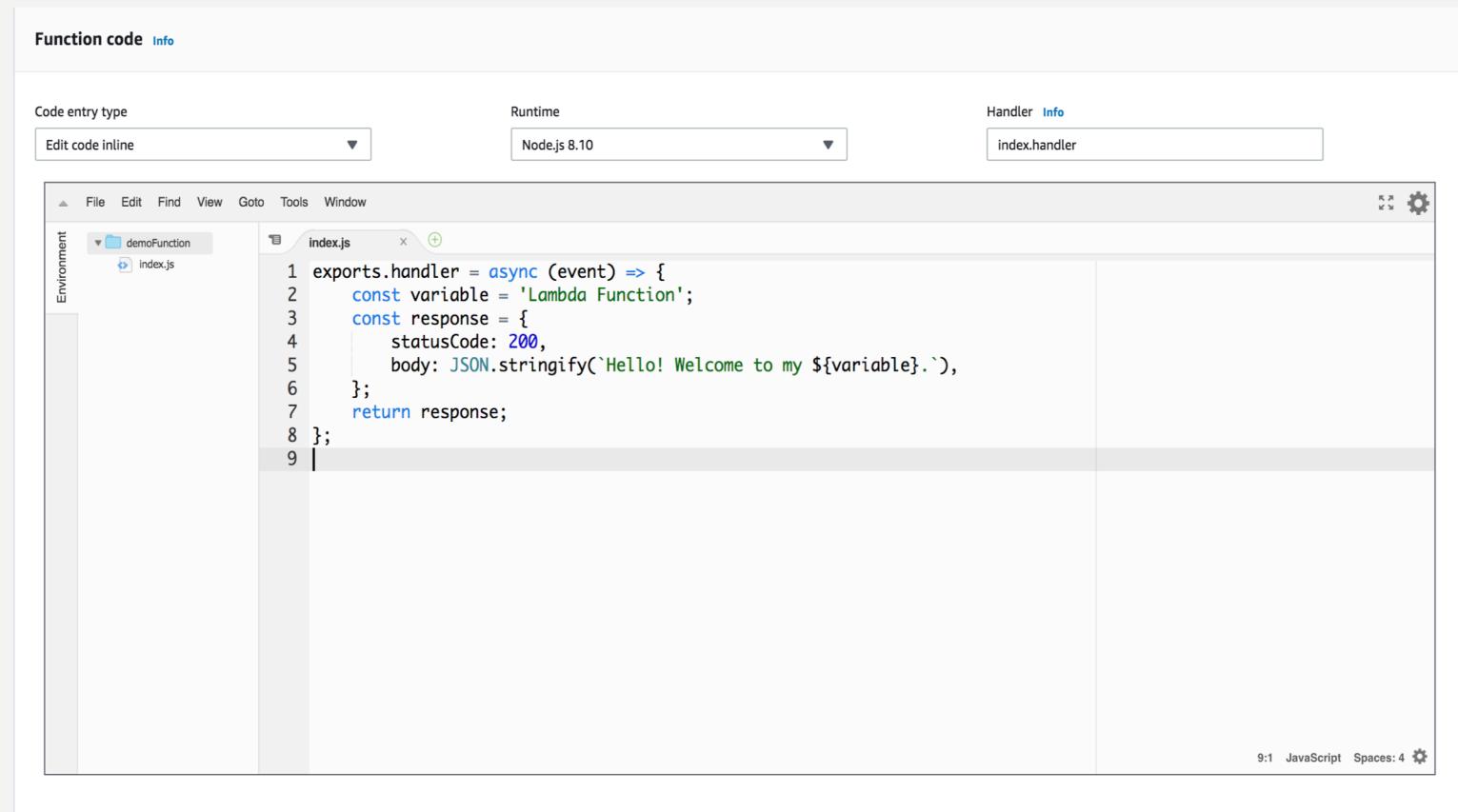
Amazon CloudWatch Logs

Resources that the function's role has access to appear here

The screenshot shows the AWS Lambda function configuration page for 'demoFunction'. At the top, a green success message states: 'Congratulations! Your Lambda function "demoFunction" has been successfully created. You can now change its code and configuration. Choose Test to input a test event when you want to test your function.' Below this, the function name 'demoFunction' is displayed along with its ARN. A navigation bar includes links for Throttle, Qualifiers, Actions, and a dropdown for 'Select a test event' with options 'Test' and 'Save'. The main area is titled 'Designer' and features a sidebar with a 'Triggers' section listing various AWS services like API Gateway, AWS IoT, Alexa Skills Kit, etc. The main workspace shows the function itself with a placeholder for triggers ('Add triggers from the list on the left') and a connection to Amazon CloudWatch Logs. A note at the bottom indicates where resources the function's role has access to will appear.

SCROLL DOWN TO THE IDE AND LET'S DIVE INTO THE CODE

- To the right, we have a bit of Node code
- AWS Lambda invokes your Lambda function via a handler object.
A handler represents the name of your Lambda function and serves as the entry point that AWS Lambda uses to execute your function code.
- Using “async” means that this function is not going to hold up any other code and will run when triggered
- When the function is triggered, we return a JSON response with a status of 200 – OK



The screenshot shows the AWS Lambda Function code editor interface. At the top, there are tabs for 'Function code' and 'Info'. Below that, there are dropdown menus for 'Code entry type' (set to 'Edit code inline'), 'Runtime' (set to 'Node.js 8.10'), and 'Handler' (set to 'index.handler'). The main area is a code editor window titled 'index.js'. The code is as follows:

```
1 exports.handler = async (event) => {
2     const variable = 'Lambda Function';
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify(`Hello! Welcome to my ${variable}.`),
6     };
7     return response;
8 };
9 |
```

The code defines an 'exports.handler' function that takes an 'event' parameter and returns a JSON response with a status code of 200 and a body containing a welcome message and the value of the 'variable' constant.

KEEP THE SETTINGS AS IS (128MS AND 3 SECOND TIMEOUT)

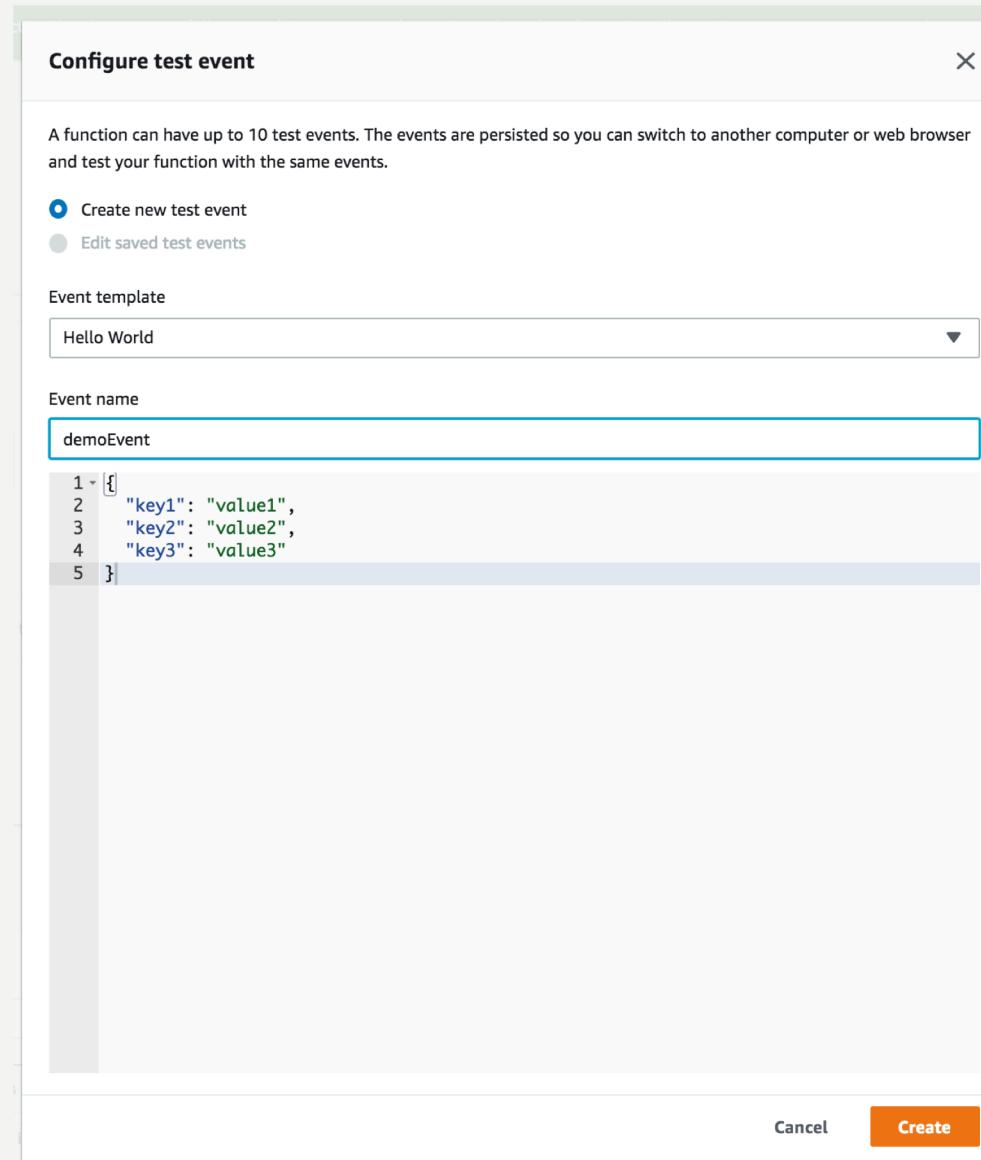
Timeout – You pay for the AWS resources that are used to run your Lambda function. To prevent your Lambda function from running indefinitely, you specify a timeout. When the specified timeout is reached, AWS Lambda terminates execution of your Lambda function.

Concurrency - The concurrency limit determines how many function invocations can run simultaneously in one region. The limit applies to all functions in the same region and is set to 1000 by default.

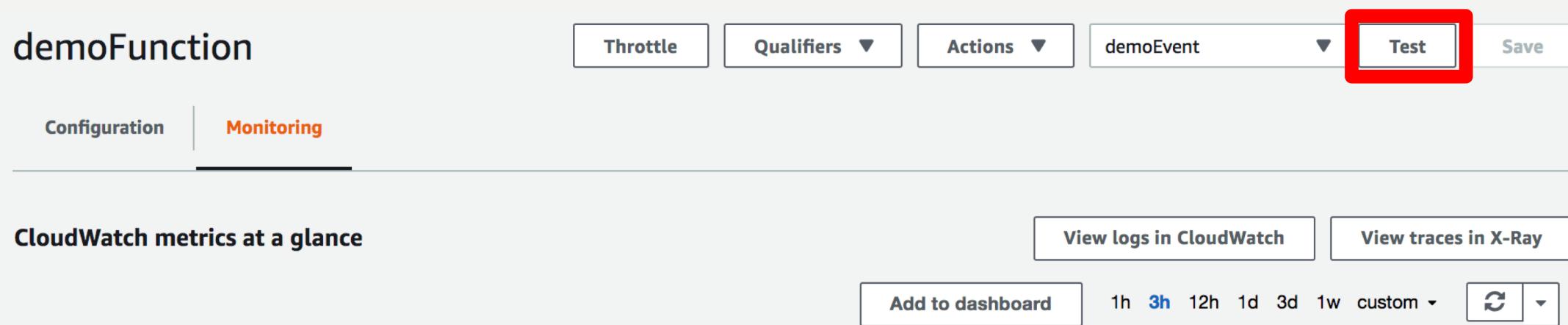
The screenshot shows the AWS Lambda Function Configuration page with the following settings:

- Execution role:** Set to "Use an existing role" with "service-role/demoFunction-role-ebql3nh2" selected.
- Basic settings:** Description is empty. Memory (MB) is set to 128 MB. Timeout is set to 3 seconds.
- Network:** Virtual Private Cloud (VPC) is set to "No VPC".
- Debugging and error handling:** DLQ resource is set to "None". Enable active tracing is unchecked.
- Concurrency:** Unreserved account concurrency is set to 1000. Use unreserved account concurrency is selected.
- Auditing and compliance:** AWS CloudTrail logging is disabled.

CREATE AN EVENT TO TEST YOUR FUNCTION



CLICK ON THE “TEST” BUTTON NEXT TO YOUR SELECTED EVENT



VIEW YOUR SERVERLESS FUNCTION RESULTS

demoFunction

Throttle Qualifiers Actions demoEvent Test Save

Execution result: succeeded ([logs](#))

▼ Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{  
  "statusCode": 200,  
  "body": "\"Hello from Lambda!\""  
}
```

Summary

Code SHA-256	Request ID
4qpUL2sqH79KqI0HwgsjQL8IDg0h98YFNTKUewLQ2lk=	b7389628-4b2c-4537-8f15-0fa8f738a518
Duration	Billed duration
28.89 ms	100 ms
Resources configured	Max memory used
128 MB	71 MB

Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: b7389628-4b2c-4537-8f15-0fa8f738a518 Version: $LATEST  
END RequestId: b7389628-4b2c-4537-8f15-0fa8f738a518  
REPORT RequestId: b7389628-4b2c-4537-8f15-0fa8f738a518 Duration: 28.89 ms Billed Duration: 100 ms Memory Size:  
128 MB Max Memory Used: 71 MB
```

TRIGGERING YOUR FUNCTION WITH A REST CALL

Go back to the AWS console and search for and click on API Gateway

AWS Management Console

The screenshot shows the AWS Management Console homepage. On the left, there's a sidebar with "AWS services" and a "Find Services" search bar containing "api gateway". Below the search bar, "API Gateway" is listed under "Recently visited services" with the description "Build, Deploy and Manage APIs". At the bottom of the sidebar, there are links for "Lambda" and "All services". To the right, there are two main sections: "Access resources on the go" featuring the AWS Mobile App icon, and "Explore AWS" featuring a link about Amazon S3.

AWS services

Find Services
You can enter names, keywords or acronyms.

api gateway

API Gateway
Build, Deploy and Manage APIs

▼ Recently visited services

Lambda

All services

Access resources on the go

Access the Management Console using the AWS Console Mobile App. [Learn more](#)

Explore AWS

Scalable, Durable, Secure Backup & Restore with Amazon S3

Discover how customers are building backup & restore solutions on AWS that save money. [Learn more](#)

CLICK GET STARTED ON THE API GATEWAY HOME PAGE



Amazon API Gateway

Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

[Get Started](#)

[Getting Started Guide](#)

CHOOSE THE REST OPTION AND CREATE A “NEW API”

The screenshot shows the 'Create' page in the Amazon API Gateway console. At the top, the navigation bar includes the 'Amazon API Gateway' logo, 'APIs > Create', 'Show all hints', and a help icon. The main section is titled 'Choose the protocol' with the sub-instruction: 'Select whether you would like to create a REST API or a WebSocket API.' Two radio buttons are present: 'REST' (selected) and 'WebSocket'. Below this, the 'Create new API' section defines what a REST API is: 'In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.' Three radio button options are shown for creating a new API: 'New API' (selected), 'Import from Swagger or Open API 3', and 'Example API'. The 'Settings' section allows entering an 'API name*' (set to 'My API'), a 'Description' (empty), and selecting an 'Endpoint Type' (set to 'Regional'). A note at the bottom left indicates '* Required'. On the bottom right, a blue 'Create API' button is visible.

Amazon API Gateway APIs > Create Show all hints ?

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

API name* My API

Description

Endpoint Type Regional ⓘ

* Required

Create API

ENTER IN YOUR API NAME AND CLICK “CREATE API”

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

API name*

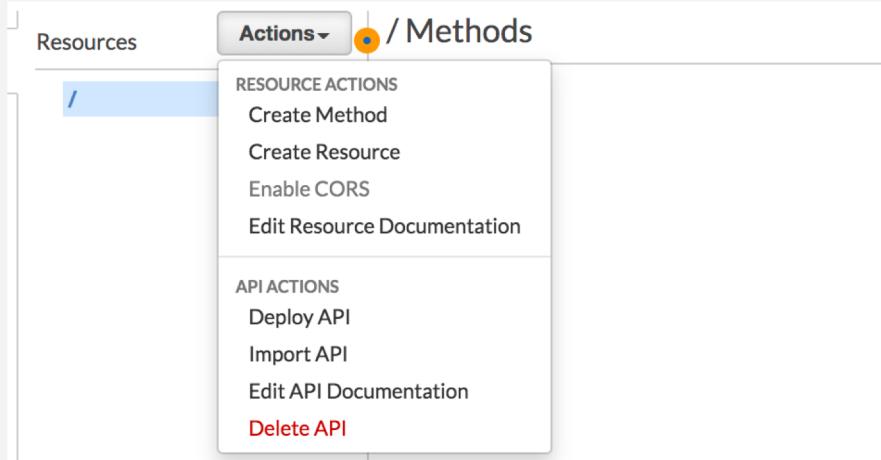
Description

Endpoint Type ⓘ

* Required Create API

LET'S MAKE A “GET” REQUEST

Step #1: Click Create Method



Step #3: Set the Integration type to “Lambda Function”

Step #4: Keep Lambda Proxy Integration empty (we will not be using a proxy)

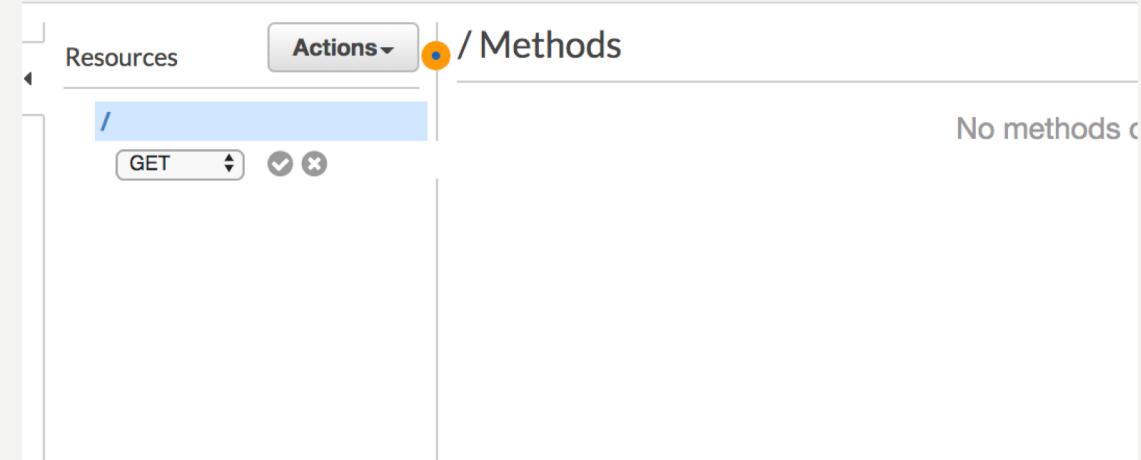
Step #5: Set Lambda region to the region where your users will be located (us-east-1 is fine)

Step #6: Set function to your function name

Step #7: Use default timeout

Step #8: Click save

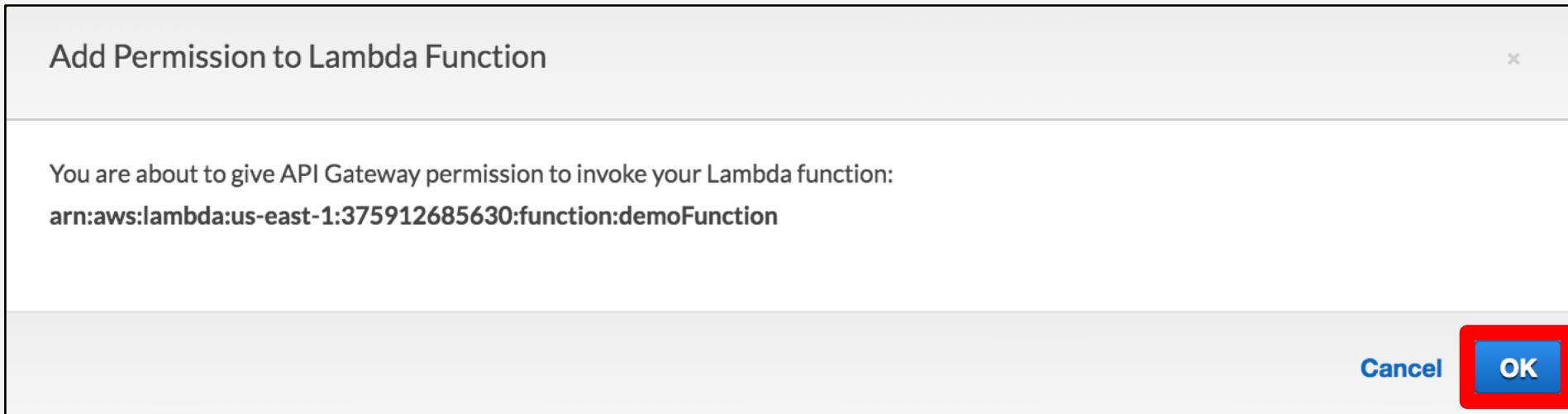
Step #2: Click on the “GET” method and check button



The screenshot shows the 'GET - Setup' configuration dialog. It has the following fields:

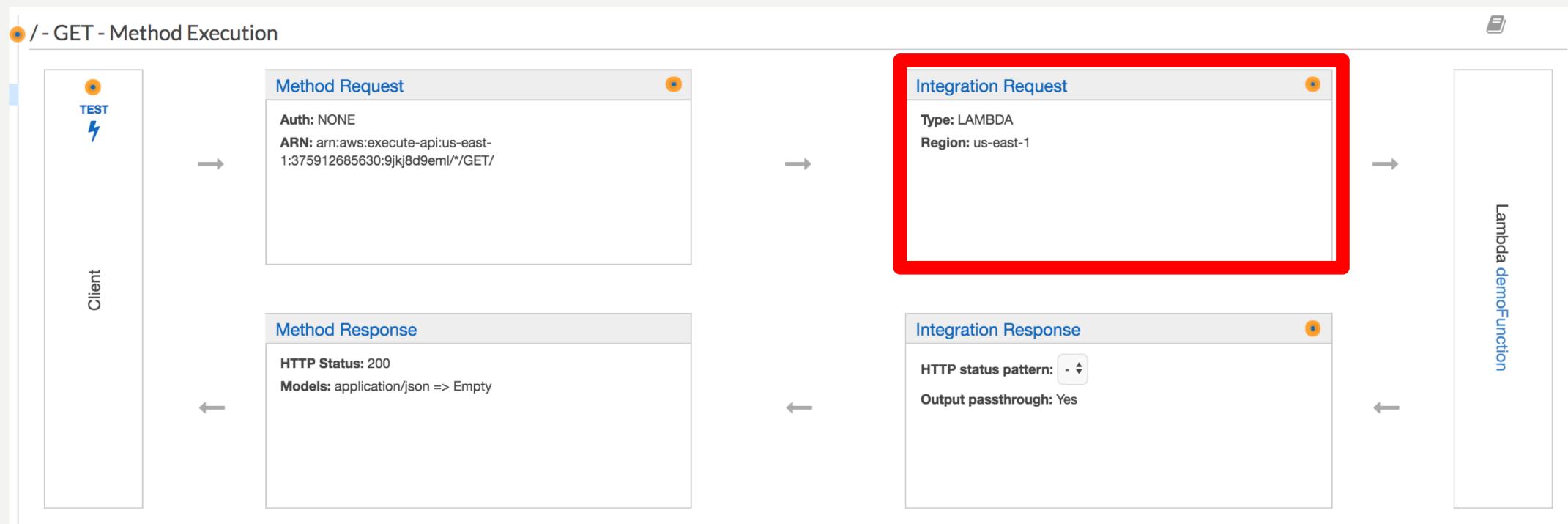
- Integration type:** Radio button selected for "Lambda Function". Other options include "HTTP", "Mock", "AWS Service", and "VPC Link".
- Use Lambda Proxy integration:** Unchecked checkbox.
- Lambda Region:** A dropdown set to "us-east-1".
- Lambda Function:** An input field containing "demoFunction".
- Use Default Timeout:** Checked checkbox.
- Save:** A blue "Save" button at the bottom right.

GIVE PERMISSION TO API GATEWAY TO CALL YOUR LAMBDA FUNCTION



SET UP THE INTEGRATION REQUEST

Click on the Integration Response configuration tile



SET UP THE INTEGRATION REQUEST

The Integration Request should be set to the following values:

Type: Lambda Function

Proxy Integration: False

Lambda Region: (the region you used for your Lambda function)

Lambda Function: (Function name)

Execution role: (empty)

Caller Credentials: False

Credentials Cache: (empty)

Default Timeout: True

The screenshot shows the 'Method Execution' configuration for a GET request. The 'Integration type' is set to 'Lambda Function'. The 'Lambda Region' is 'us-east-1' and the 'Lambda Function' is 'demoFunction'. The 'Execution role' field is empty. Under 'Invoke with caller credentials', the 'Credentials cache' option is set to 'Do not add caller credentials to cache key'. The 'Use Default Timeout' checkbox is checked. There are four expandable sections at the bottom: 'URL Path Parameters', 'URL Query String Parameters', 'HTTP Headers', and 'Mapping Templates'.

ADD MAPPING TEMPLATE

Set the “Requested body passthrough” to:“When there are no templates defined (recommended)”

Click on “Add mapping template” and enter in:“application/json”

Then click on the check mark next to the input field

Click “Save”

The screenshot shows two separate configuration sections for 'Mapping Templates'.

Top Configuration:

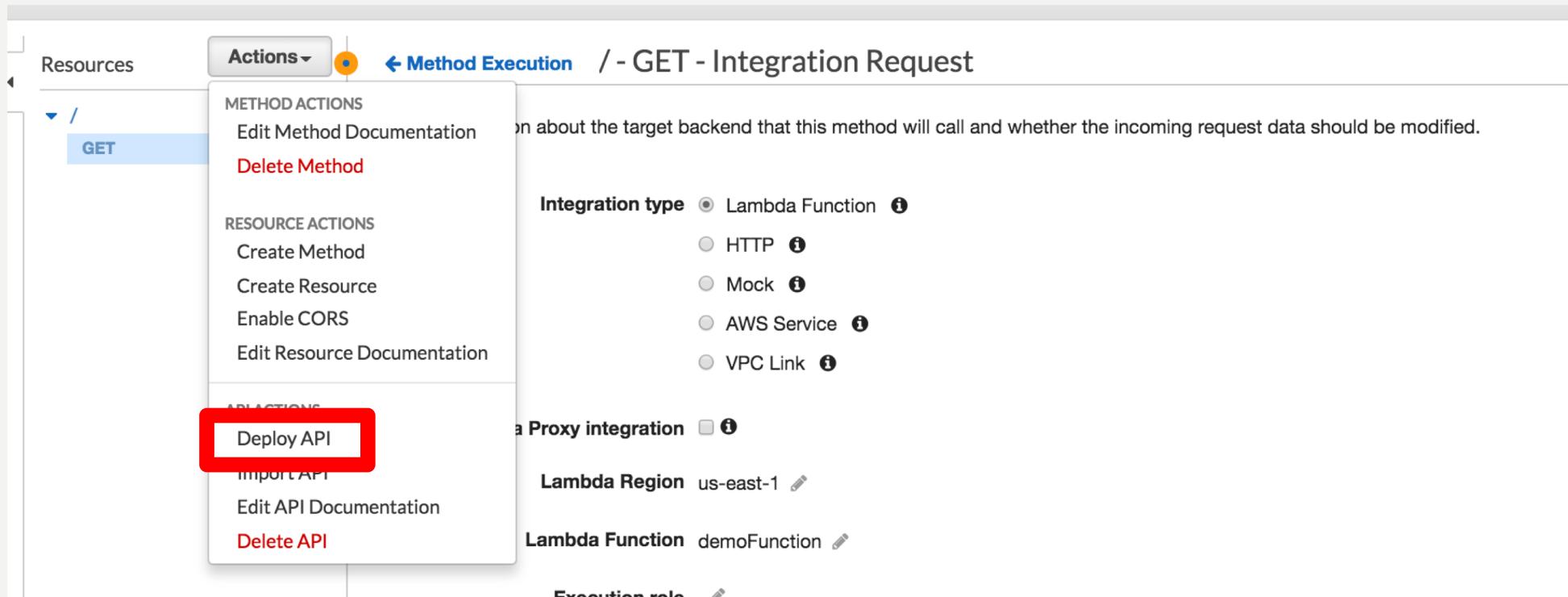
- Request body passthrough:** The radio button for "When there are no templates defined (recommended)" is selected.
- Content-Type:** A note states: "No mapping templates defined. The request body will be passed through to the integration endpoint".
- Add mapping template:** A blue button with a plus sign.

Bottom Configuration:

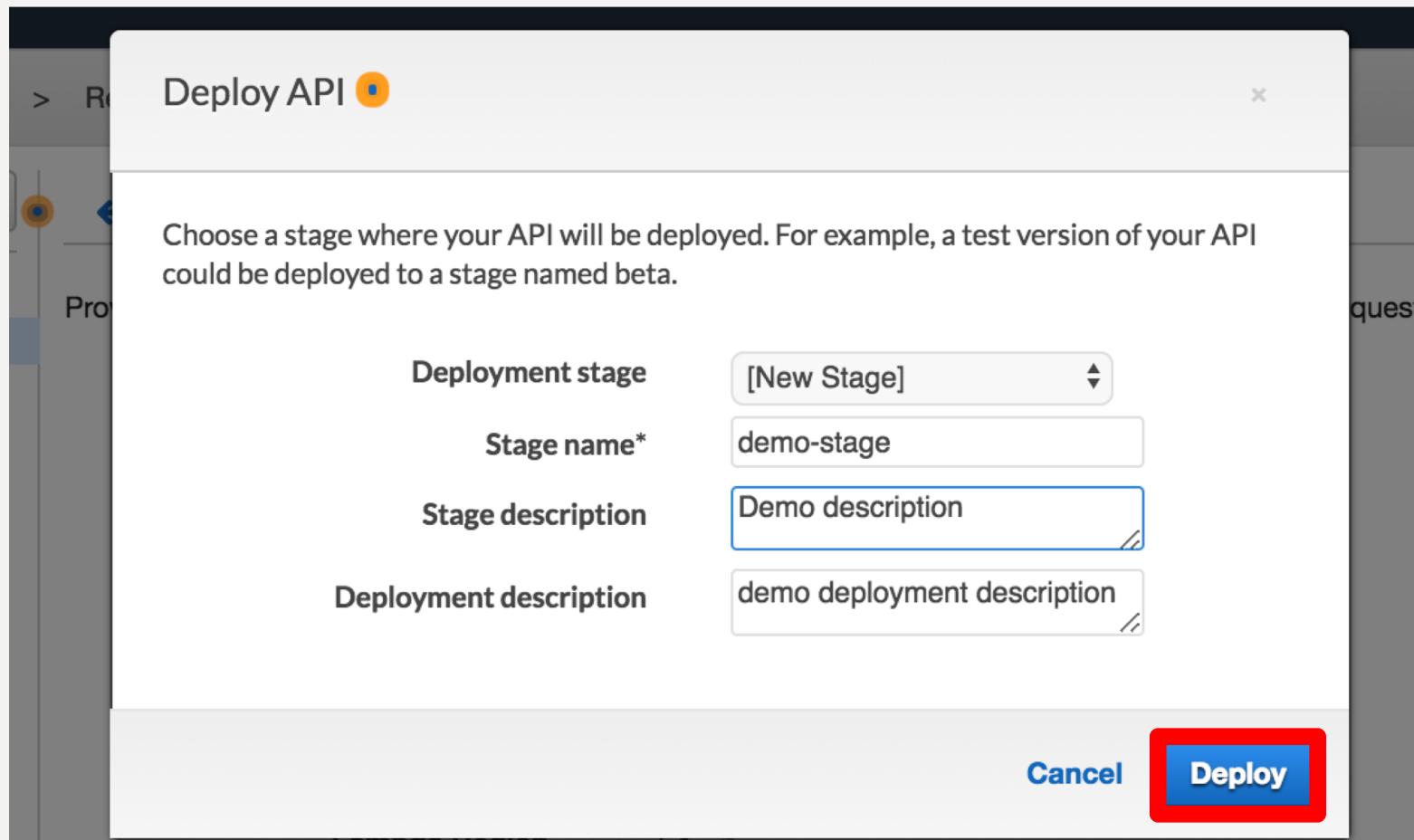
- Request body passthrough:** The radio button for "When there are no templates defined (recommended)" is selected (highlighted with a red box).
- Content-Type:** The input field contains "application/json" and has a checked checkbox to its right (highlighted with a red box).
- Add mapping template:** A blue button with a plus sign.

DEPLOY YOUR API!

Click on Deploy API on the Actions Dropdown



CREATE “NEW STAGE” WHERE YOUR API WILL BE DEPLOYED AND CLICK “DEPLOY”



CLICK ON YOUR “INVOKE URL”

The Invoke URL will be used to call your function.

demo-stage Stage Editor Delete Stage

Invoke URL: <https://9jkj8d9eml.execute-api.us-east-1.amazonaws.com/demo-stage>

Settings Logs/Tracing Stage Variables SDK Generation Export Deployment History Documentation History Canary

Cache Settings

Enable API cache

VIEW YOUR FUNCTION RESULTS

- Your results will be in JSON format which is what we selected for our integration request
- The status code of “200” stands for OKAY and means that our call was successful
- The body of the JSON shows the returned value from the function

```
{  
  statusCode: 200,  
  body: '"Hello! Welcome to my Lambda Function."'  
}
```

*Hint: to more easily read your JSON response, download the JSONView Chrome plugin which auto-formats any JSON that appears in the body (I'm using the plugin to display the above)

CONGRATS ON CREATING YOUR FIRST SERVERLESS FUNCTION!

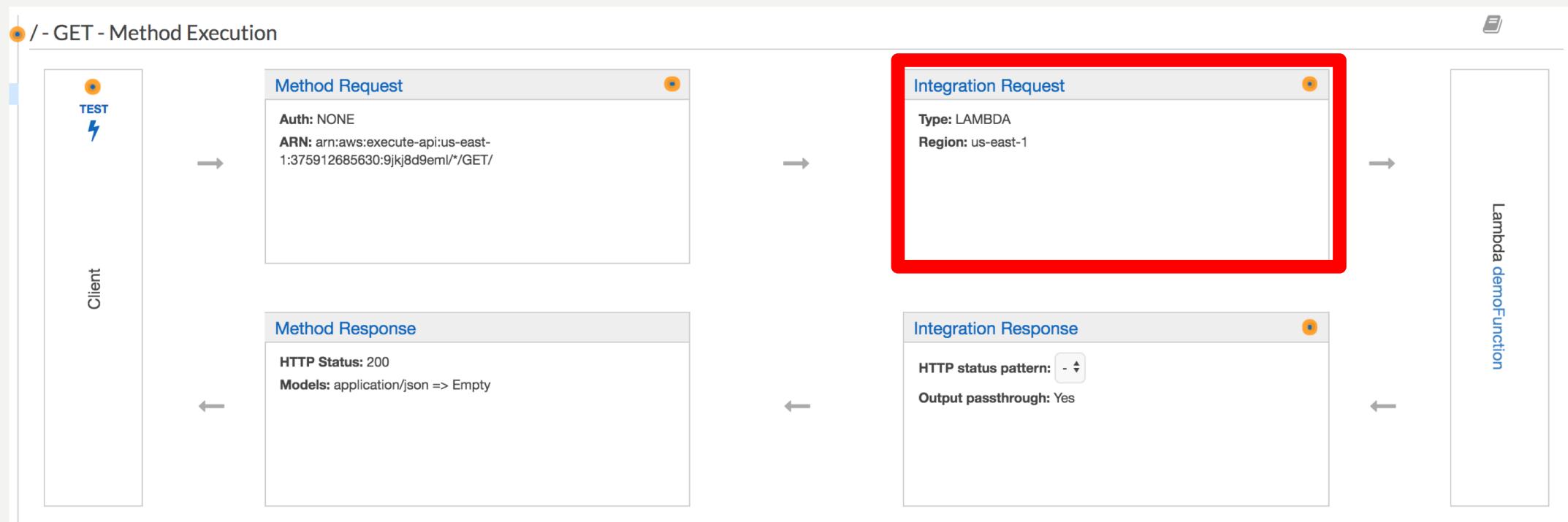
Let's try passing data into your Lambda Function.

To start, go back to API Gateway home: <https://console.aws.amazon.com/apigateway/home>

Then, click on your newly created API and then the GET request

SET UP THE INTEGRATION REQUEST

Click on the Integration Response configuration tile



SET UP THE INTEGRATION REQUEST

The Integration Request should be set to the following values (These values should already be set from the initial configuration):

Type: Lambda Function

Proxy Integration: False

Lambda Region: (the region you used for your Lambda function)

Lambda Function: (Function name)

Execution role: (empty)

Caller Credentials: False

Credentials Cache: (empty)

Default Timeout: True

The screenshot shows the 'Method Execution' configuration for a GET request. The 'Integration type' is set to 'Lambda Function'. The 'Lambda Region' is 'us-east-1' and the 'Lambda Function' is 'demoFunction'. The 'Execution role' is left empty. The 'Invoke with caller credentials' option is unchecked. The 'Use Default Timeout' checkbox is checked. Below these settings, there are sections for 'URL Path Parameters', 'URL Query String Parameters', 'HTTP Headers', and 'Mapping Templates'.

Method Execution / - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type Lambda Function i
 HTTP i
 Mock i
 AWS Service i
 VPC Link i

Use Lambda Proxy integration i

Lambda Region us-east-1 edit

Lambda Function demoFunction edit

Execution role edit

Invoke with caller credentials i

Credentials cache Do not add caller credentials to cache key edit

Use Default Timeout i

▶ URL Path Parameters

▶ URL Query String Parameters

▶ HTTP Headers

▶ Mapping Templates edit

ADD MAPPING TEMPLATE

Set the “Requested body passthrough” to:“When there are no templates defined (recommended)”

Click on “Add mapping template” and enter in:“application/json”

Then click on the check mark next to the input field

The screenshot shows a configuration interface for 'Mapping Templates'. It consists of two main sections, each with a title, a radio button group for 'Request body passthrough', a 'Content-Type' section, and an 'Add mapping template' button.

Top Section:

- Title:** Mapping Templates (with a blue info icon)
- Request body passthrough:** Radio buttons:
 - When no template matches the request Content-Type header (info icon)
 - When there are no templates defined (recommended) (info icon)** (This option is selected and highlighted with a red box.)
 - Never (info icon)
- Content-Type:** No mapping templates defined. The request body will be passed through to the integration endpoint
- Add mapping template:** A blue button with a plus sign and a 'Content-Type' dropdown menu.

Bottom Section:

- Title:** Mapping Templates (with a blue info icon)
- Request body passthrough:** Radio buttons:
 - When no template matches the request Content-Type header (info icon)
 - When there are no templates defined (recommended) (info icon)** (This option is selected and highlighted with a red box.)
 - Never (info icon)
- Content-Type:** application/json (highlighted with a blue border and a red box around the checkmark icon)
- Add mapping template:** A blue button with a plus sign.

ADD MAPPING JSON

Add a mapping from the variable you wish to use in your lambda function to the same variable name that you will be placing on the URL as a parameter.

Example:

URL: .../demo-
stage?demoVariable=test

In the above example, demoVariable would be our variable name.

Click “Save” once you have added your mapping.

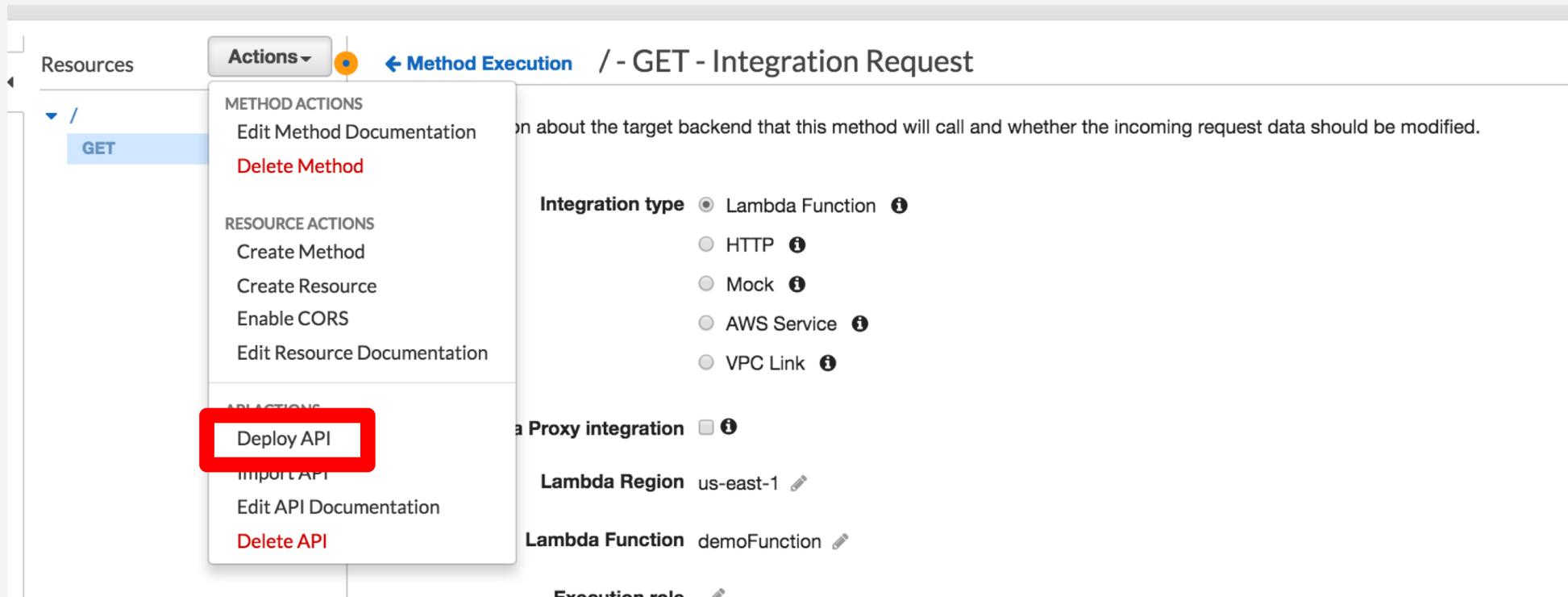
The screenshot shows the 'Mapping Templates' section of the AWS Lambda function configuration. It includes a dropdown for 'Request body passthrough' (set to 'When there are no templates defined (recommended)'), a 'Content-Type' field set to 'application/json', a 'Generate template:' dropdown, and a code editor containing the following JSON template:

```
1 {  
2   "demoVariable": "$input.params('demoVariable')"  
3 }
```

At the bottom right are 'Cancel' and 'Save' buttons.

DEPLOY YOUR API!

Click on Deploy API on the Actions Dropdown



RETRIEVE THE URL PARAMETER VALUE FROM WITHIN YOUR LAMBDA FUNCTION

Now that we have the URL parameter based argument set up in our API gateway, let's use that variable in our Lambda Function.

To start, go back to Lambda home: <https://console.aws.amazon.com/lambda/home>

Then, click on your newly created function

MODIFY THE LOGIC TO RETRIEVE THE VALUE FROM THE EVENT

Pull the value you set up on the API Gateway from the event object

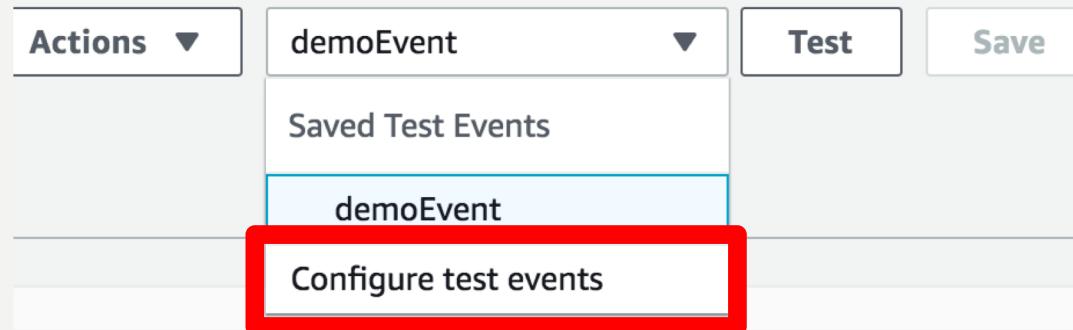
The screenshot shows the AWS Lambda function configuration interface. At the top, there are three dropdown menus: 'Code entry type' (set to 'Edit code inline'), 'Runtime' (set to 'Node.js 8.10'), and 'Handler' (set to 'index.handler'). Below these, the function name 'demoFunction' is shown, along with the file structure 'index.js'. The main area displays the 'index.js' code:

```
1 exports.handler = async (event) => {
2     const variable = event.demoVariable;
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify(`Hello! Welcome to my ${variable}.`),
6     };
7     return response;
8 };
9
```

The code defines an asynchronous handler named 'handler' that takes an 'event' parameter. It retrieves a value from the 'demoVariable' key in the event object and constructs a response object with a status code of 200 and a body containing a personalized message.

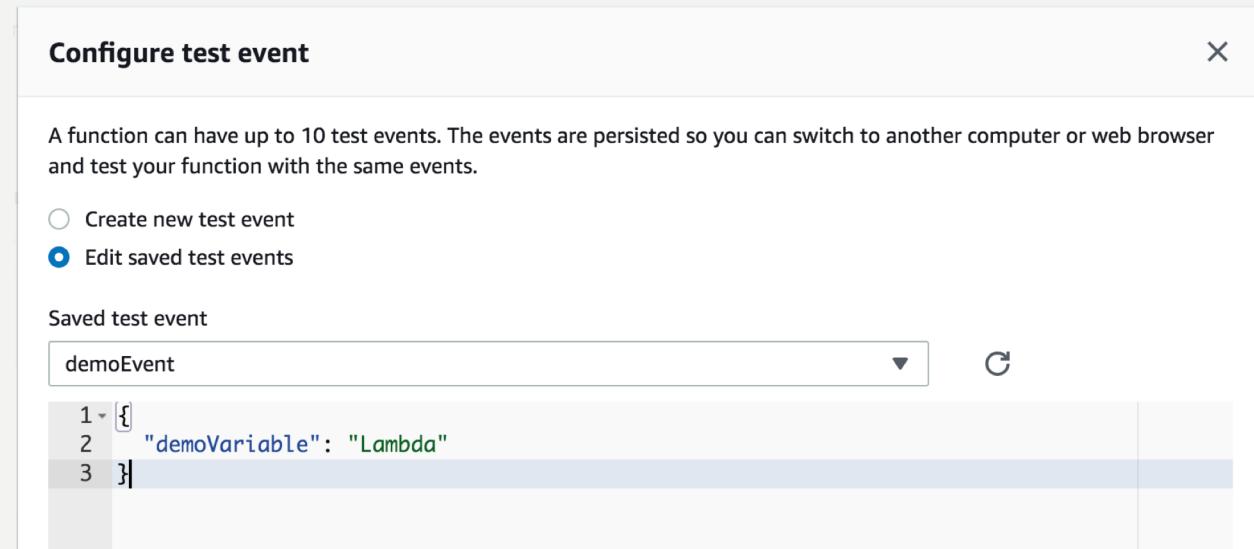
UPDATE YOUR LAMBDA FUNCTION TEST DATA

Click on "Configure test events"



Modify the data and set your variable name to an expected value

Once the value has been set, click save.



TEST YOUR LAMBDA FUNCTION WITH PASSED IN DATA

demoFunction

Throttle Qualifiers Actions demoEvent ▾ Test Save

Execution result: succeeded ([logs](#))

▼ Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{  
  "statusCode": 200,  
  "body": "\"Hello! Welcome to my Lambda.\""  
}
```

Summary

Code SHA-256	Request ID
R3HjdLt0iZ1WxNlxc9qXkoLPbVNwHlk80cYYeEb0e4Y=	3d731cc7-6598-46c6-a198-1ae31c6e35eb
Duration	Billed duration
18.59 ms	100 ms
Resources configured	Max memory used
128 MB	72 MB

Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: 3d731cc7-6598-46c6-a198-1ae31c6e35eb Version: $LATEST  
END RequestId: 3d731cc7-6598-46c6-a198-1ae31c6e35eb  
REPORT RequestId: 3d731cc7-6598-46c6-a198-1ae31c6e35eb Duration: 18.59 ms Billed Duration: 100 ms Memory Size: 128 MB  
Max Memory Used: 72 MB
```

TEST YOUR UPDATED LAMBDA FUNCTION WITH THE API GATEWAY

When you test your function with the the variable use in the URL Path, you should see that URL variable being used in the body of the response



A screenshot of a web browser window. The address bar shows a secure connection to the URL: `https://9jkj8d9eml.execute-api.us-east-1.amazonaws.com/demo-stage?demoVariable=lambda%20function%20called%20from%20an%20API%20Gateway`. The page content displays a JSON response:

```
{  
  statusCode: 200,  
  body: '"Hello! Welcome to my lambda function called from an API Gateway."'  
}
```

FOR MORE INFORMATION:

Building Lambda Functions:

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-app.html>

Creating a REST API in Amazon API Gateway:

<https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-create-api.html>

Amazon API Gateway API Request and Response Data Mapping Reference:

<https://docs.aws.amazon.com/apigateway/latest/developerguide/request-response-data-mappings.html>

THANK YOU!